CS170 Project Reflection
Aaron Soll
James Holcombe
Brendan Hogg

Our final and most successful approach was a dynamic programming solution that leveraged a simpler subproblem, where profit is equal to perfect benefit if done before the deadline and zero if done after the deadline. We first created a helper function which solved this subproblem, made possible by the fact that each task must appear in increasing order of deadline (or not at all). This helper function first sorted tasks by deadline, then tested them one by one to see if they would improve performance. For each task, if there was not enough room to add it to the existing list before its deadline, the worst tasks (in terms of profit per duration) would be temporarily *popped off* the list until the new task could fit; if the new arrangement was better, the new task would stay. Otherwise, the popped off tasks would be placed back in the list. Once this helper function was created, we ran it on the input files after first creating many copies of each task, each offset by 1 timestep. The solver was also constrained so that no two copies of the same task could be added. For example, a task with a perfect benefit of 10 and a deadline of 100 would have 1340 copies made; one with a deadline of 101 and perfect benefit of 10*e^(-.017 * 1), one with a deadline of 102 and a perfect benefit of 10 * e^(-.017 * 2), etc. The algorithm would output the optimal ordering from this discretized version of the problem. This algorithm worked well because it only added each task at each timestep if it actually improved the total benefit, but still allowed for each task to be removed later on. Greedy algorithms, on the other hand, lack the ability to remove tasks later on. Furthermore, this approach was strong because it used a pretty accurate heuristic-- profit per unit duration. When popping off tasks from the list to try a new one, it is likely that we are popping off the right ones, since they are the ones with the lowest overall value.

Before we reached this approach, we came up with one much worse algorithm. This algorithm iteratively calculated each task's "score" based on a priority function, added the best one to a list, and repeated until all tasks had been completed or no more could be added. The priority function simply calculated a linear combination of price/duration and (-deadline). For most of the outputs, this approach only got around 60-80% of the total benefit of the top submissions, so it did not perform very well at all. As a result, we started thinking about the DP type solutions that we ended up using for our final algorithm.

We ran all our trials on our personal computers, and did not use any other computational resources during the project.